Tampere University

Tomi Summanen

HIERARCHICAL STRUCTURE OF AUTOMATION APPLICATION

Faculty of Engineering and Natural sciences
Master of Science Thesis
October 2020

ABSTRACT

Tomi Summanen: Hierarchical structure of automation application
Master of Science Thesis
Tampere University
Master's Degree Programme in Automation Technology
October 2020

The automation application that controls a process is designed separately for each plant. The plant specific application development is based on initial information such as functional descriptions and diagrams. An automation engineer transforms those specifications into automation application with the features the automation system offers.

The applications are designed as function block diagrams. The function block language has a limited capability to express complex functions so thousands of diagrams are needed to represent the full functionality of a factory. The application is built of separate device specific controls that are connected to form the control logic of the factory. The automation applications do not conform to a uniform format as all suppliers have their own.

Designing automation application is a multiphase and multi-stakeholder process. The application engineering is time consuming and the used structures are inefficient. The engineering can be accelerated by using former applications, either productized solutions or copying existing functionality.

This thesis concentrates on alternative application format. A hierarchical format where the level of application details varies between design levels was proposed to streamline engineering. The focus was to assemble a model that would fit into continuous process automation. The main research method was user interviews. The interviews are supported by review of the application engineering process and its effecting variables.

Current application model contains too much variation and is too detailed for intended use. The main development item towards better format is the possibility to use application diagrams as visual design elements in other diagrams. An abstract format would allow the design of larger structures. Other implementation details and requirements came up that are worthwhile to consider if changing the application structure. The hierarchical structure was supported by its ability to adapt to the needed detail level and to visualize the structure better. The proposed model would not on its own help the application engineering efficiency if there is no change on the specification format.

Keywords: Automation application, hierarchical design

TIIVISTELMÄ

Tomi Summanen: Automaatiosovelluksen hierarkkinen rakenne
Diplomityö
Tampereen Yliopisto
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Lokakuu 2020

Prosessiautomaatiossa sovellukset, jotka ohjaavat tehtaan toimintaa suunnitellaan usein tehdaskohtaisesti. Tehdaskohtaisen automaatiosovelluksen kehitys pohjautuu lähtötietoihin kuten toimintakuvauksiin ja -kaavioihin. Automaatiosuunnittelija muuntaa kuvatun toiminnon automaatiojärjestelmän ominaisuuksien puitteissa sovellukseksi, jolla prosessia ohjataan.

Automaatiosovellukset suunnitellaan toimilohkokaavioina. Toimilohkokaavioiden ilmaisuvoima on rajallinen ja niitä tarvitaan tehtaan toiminnallisuuden kattamiseen tuhansia. Sovellus rakentuu laitekohtaisista toimilohkokaavioista, jotka yhteen kytkettyinä muodostavat koko prosessin ohjauksen. Automaatiosovellukset eivät noudata yhtenäistä rakennetta vaan eri toimittajilla on omat mallinsa.

Automaatiosovelluksen suunnittelu on monivaiheinen ja useampaa eri toimijaa koskeva kokonaisuus. Suunnittelu on aikaa vievää ja käytetyt rakenteet epätehokkaita. Sovelluksen suunnitteluun kuluvaa aikaa voidaan vähentää käyttämällä valmiita ratkaisuja joko tuotteistettujen tai kopioitujen rakenteiden muodossa.

Diplomityössä kartoitetaan sovellusrakenteen muuttamista hierarkkiseen malliin. Hierarkkisella eli allekkaisten eri tarkkuustason omaavien rakenteiden avulla haetaan suunnitteluun tehokkaampia työtapoja. Työn tavoitteena oli saavuttaa näkemys hierarkkisesta sovellusrakenteesta, joka sopii jatkuvatoimiseen prosessiautomaatioon. Selvitys on tehty pääasiassa käyttäjähaastatteluin. Haastatteluiden tueksi työ avaa suunnitteluprosessia ja siihen vaikuttavia tekijöitä, joita tukee kirjallisuuslähteet sekä muiden sovellusrakenteiden selvitys.

Nykyisen automaatiosovelluksen mallin todettiin sisältävän liikaa vaihtelua ja sen sisältö paljastaa liikaa yksityiskohtia. Oleellisimpana muutoksena kohti parempaa sovellusmallina on abstraktimpi toimilohkokaavio, joka mahdollistaa muiden toimilohkokaavioiden käytön suunnitteluelementteinä. Abstraktimmalla mallilla, jossa on mahdollista käyttää suurempia kokonaisuuksia, voitaisiin suunnitella laitekokonaisuuksia ohjaavia rakenteita. Työssä nousi esiin vaatimuksia ja yksityiskohtia, jotka on hyvä huomioida, jos rakennetta muutetaan. Hierarkkista rakennetta tukee sen mahdollisuus työtehtävään sopivaan yksityiskohtien määrään sekä visuaaliseen rakenteen hahmottamiseen. Esitetty sovellusmalli sellaisenaan ei nopeuta suunnittelutyötä, jos lähtötietoihin ei saada rakenteellista muutosta, päinvastoin sillä voi olla negatiivinen vaikutus.

Avainsanat: Automaatiosovellus, hierarkkinen suunnittelu

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

IEC International Electrotechnical Commission
ISA International Society of Automation
P&ID Piping and instrumentation diagram
VGB Technical association of energy plant operators
UML Unified modeling language

.

1. INTRODUCTION

Complexity of automation applications in industrial control systems is growing with the increased calculation capabilities and demand for more refined controls. An industrial process control application is the part of the software that controls the functionality of the machine or process by measuring and influencing the process variables. The structure of an automation application describes what kind of program parts it consists of and what kind of entities it is grouped into.

The automation application is created as a project that answers to a specific need. Usually these industrial applications are programmed using IEC 61131-3 (International Electrotechnical Commission) based programming languages. The application consists of visually presented diagrams of logical functionalities or lists of commands. These individual applications are linked together to make the process operate.

Engineering the project specific solution on detail level is time-consuming. Concepts such as composite functionalities can be used to structure the application and simplify the overall picture. The concepts used lack the fines of normal programming languages since engineers are not usually programmers. This results in ill-formed application that is tailored for the specific case. The complex structure affects directly to engineering costs and indirectly to training needs.

This thesis addresses the problem of complex application structure in large scale automation system. A hierarchical structure is analysed as an option for current solutions. In this thesis the hierarchical application model refers to application that can be viewed in separate levels having different amount of details. The basic idea of hierarchical application structure is a pyramid where on the top levels the whole automation application is simplified and when stepped down the levels the details of the application are visualized. Such an approach is used in system design, but it is not transmitted to the application.

The idea of this structure is not anything new and is proposed earlier [1]-[4], but the industry has not adopted it. The idea is to review the possibilities and the main aspects of hierarchical application structure in process automation domain to simplify the application engineering. To support adaptation of this model in industry the focus is on engineering the application.

The goal of this thesis is to present a better way of structuring the automation application. The sought outcome is a proposal of how to build hierarchical automation applications that support efficient engineering. The work strives to find the problems of design process and the reasons behind these.

The empirical part of the thesis consists of user interviews. Automation experts in various positions are interviewed on the key aspects in application engineering and application structure. This part of the work leans towards Valmet DNA as all interviewees are familiar with it. A sub process of pulp mill is drafted using the idea of the hierarchical application structure to support the interviews.

In the second Chapter the design phases of which automation application engineering is part of are shortly introduced. The third Chapter goes through basic structure and engineering of automation application to familiarize the reader on the basic concepts. That is followed by related standards and guidelines along with earlier studies. A sub process was modeled and presented in Chapter 5 to help the interviewees understand the concept. The research methods, semi-structured interview and literature review are described in Chapter 4 and the interviews themselves in Chapter 6. Chapter 7 contains the results of the interviews.

2. BACKGROUND

2.1 Automation project

Development of automation application is a part of a larger project. Such a project aims to design and implement an automated system that fulfils the requirements of the stakeholders. A systematic approach is required to verify a successful execution of the project. This process is often referred as systems design.

A system goes through several phases during its life cycle. Different names, amount and division of the content for the phases exist. The following division is used for this work: requirements analysis, specification, design, implementation, operation and retirement. This division to phases helps to create a framework to ensure the system to meet the requirements throughout its life. [5], [6]

A sequential approach provides a framework for a project consisting of multiple companies and peoples. The design phases follow each other as the design progresses, attention is given to the documentation and verification of the phases. Several approaches such as waterfall, spiral or V-model exist for orchestration of the design phases. [5], [7]

System design starts with defining the top-level requirements. Requirements analysis tries to answer to what is to be achieved. Requirements consist of different level requirements, overall mission and stakeholder requirements are refined more into system and component requirements. Requirements enable verifying the design during qualification. [7]

The requirements are refined into specifications that answer how something should be done. As with the requirements analysis it is an iterative process that happens on different levels of detail. Moving closer to the implementation phase the details get more precise.

Design phase produces details that the specifications are lacking to implement the system. Details are produced for all aspects such as electrical, mechanical and software. If the specification is detailed enough there might be no need for further design.

Implementation phase is where the system is manufactured and coded according to the requirements and detail specifications. The specific components are tested individually and integrated into the system. An acceptance test is performed to verify the functionality and the final documentation is created. [6]

Operation phase is where the system meets its purpose and produces the services it was designed for. Upgrades, maintenance and modifications are carried out in this phase. These actions require the understanding of the existing implementation and design.

In retirement stage the system is removed from operation. Activities for this stage relate to the safe disposal of the system. Planning of the retirement is part of the early stages of system lifecycle design as awareness of material disposal is increasing. [5]

2.2 Design architecture

The decomposition process is defining the functional and physical architectures based on the requirements and refining the system requirements into component requirements. These form the hierarchical model of the system. Functional hierarchy contains the functions performed by the system and its components. And the physical architecture contains the resources of which the system is built. One to one mapping between these is often the best solution [7]. Following paragraphs considers the hierarchy from automation application perspective.

The system is divided to smaller systems and system elements until those cannot be further decomposed or the elements are defined in such detail that those can be implemented [5]. Discipline engineers are to design the items the bottom layer defines [7]. The development process of a functional hierarchy is discussed in more detail in [7].

The amount of levels depends on the complexity of the interactions in the application. It also depends on the guidelines that are followed, from 3 [8] to 7 [9] levels are suggested. In [8] those division levels are from bottom to up: individual control area, group control area and process control area. The lowest level unites the functions of any device that are in direct connection of the process trough detection and actuating. The group level unites the functions that governs a section of the individual level and the highest level covers the functions that govern the group level. Levels can be left out if needed but individual level should be always available, with some exceptions [8].

IEC 81346-1 [10] defines general principles of structuring industrial automation system information and designation codes. It divides objects into structures for manageability reasons. It defines three aspects for the code system: function, location and product. And all objects need to be part of at least one of the aspects. An object has a designation code based on the view that is required. The code system builds hierarchically describing the full path to the object, for example the location hierarchy may start from the room it

is located, then the cabinet assembly, and the location in the cabinet. [11] The designation code should propagate through the application structure into the sub functionalities automatically.

The hierarchy should address the functionality in various modes such as error state, normal operation or partial operations [7]. Categorizing the system functions may help the decomposition process. Automation related functions can be categorized following [8] to measuring, actuating, control, protection, automatic control and monitoring.

Measuring task covers detection and preparation off process data and linking the data to control functions. Actuating task covers individual level functional diagrams that contain the information of equipment controls. Control task displays the automated steps of the process, this might be better understood as sequences or procedural controls. Protection task contains the controls related to safety regarding humans and assets. Automatic control task presents the automatic controls and control algorithms in a way that shows the associations within the process section. Monitoring task's purpose is to display and record process values and signal for limit value violations and status changes for the purpose of fault analysis.

These tasks have different requirements on how specific the specifications are for implementation. Tasks that relate straight to the physical devices require the decomposition process of functional hierarchy to reach the actuating device. For procedural controls, protection and automatic controls the functional hierarchy need not be that specific. Guideline [8] requires the details to be present on the most elemental level.

2.3 Application engineering

Activities regarding the application engineering start in the specification phase of the system engineering where an automation system supplier is selected. Data of the system scope is needed to make the selection. Important factors for the automation system are price, comprehensibility, changeability, extensibility, quality and traceability [12]-[16].

Developing automation application (Chapter 3.1) is part of the implementation phase. It follows the functional requirements and detail specifications produced by earlier stages and works in close contact with physical design. An application engineer translates the knowledge specified by process engineer to the automation system to control the process.

Before the system is taken into operation several approval phases for the application and the integrated system are accomplished. The scope of the tests is specified in earlier

phases of design, those can contain testing to “design to specification” or qualitative aspects of the system such as recovery from a power failure. System documentation is also provided at the end of the implementation.

Operational phase of the automation application is similar as for the rest of the system, changes are made if needed and the application can be further modified by maintenance personnel. Application life cycle might come to its end before the rest of the system, caused by the faster development of control systems and electronics. In such case the system is reinstalled with new automation system.

2.4 Models

Models have been used in system design activities for decades. The first models were used to decomposition the system functions, connecting the system elements and modeling the dynamic behaviour. Block diagrams were used to model system components and the exchange of information or physical entities between those. [7]

Model is an abstract representation of the system or its element. It contains the relevant information of the subject such as the high-level requirements or detailed physical and mathematical aspects. Models are used in the system design process by various engineering disciplines [5], [7].

Model can be understood as diagrams depicting the overall functionality. Models are not the same as drawings. Model can be represented as a drawing, but a drawing or a diagram is not a model. Figure 1 illustrates the difference, model is data that you can view in several formats of from different angles but the hard copy document that the automation field uses is just a snapshot of the data from a certain angle.

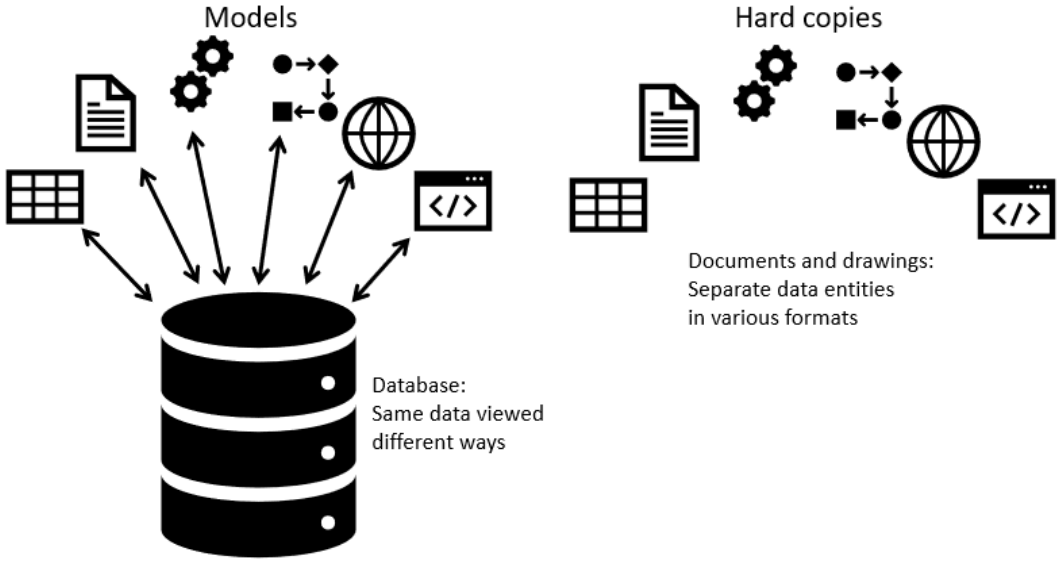

Figure 1. *Models are an image of different angles of the same data or generated of it.*

Model based engineering

To differentiate model-based engineering from traditional engineering the traditional engineering could be called document centric. In document centric design models may be used but those are separate and supplemented by other definition documents and lists. In model-based approach this information is captured as far as possible into a model of the system. [5]

Model transformation is a process where data defined in another format is transformed into another based on a set of rules. Model transformations are a key aspect of model-based engineering. Transformations are used to model the data from different viewpoints to reduce complexity. [17]

Modeling languages

Model based engineering is a methodology it does not itself specify the notation, therefore several modeling notations have been developed, UML (unified modeling language) being one of the most known. Currently going on version 2.5.1 from 2017 its been there for over 20 years already. [18] It is managed by the object management group OMG, they describe it as “A specification defining a graphical language for visualizing, specifying, constructing, and documenting the artefacts of distributed object systems.” It consists of 13 diagram types divided into three categories.

As UML is not specific to automation field it may lack the details that are required in that area. For such uses it can be expanded or restricted using profiles. One of those is

SysML that is designed for modeling systems. It restricts the amount of diagram types and adds system design specific ones such as requirement and parametric diagram types.

Another profile that fits into the automation field is UML AP automation profile which is explained in more detail in [19]. It adds elements from SysML and basic UML. Those new elements bring more familiar terminology for the process and automation design.

The lack of using models in automation domain may be due to the professionals being accustomed to using low-level constructs in application engineering and the used modeling languages are complex and too general for use in process control [20]. According to [6] the process controls are not modeled with the system, but the work is started after the system has been modeled.

3. AUTOMATION APPLICATION

A modern automation system is connected to the enterprise control system. Figure 2 helps to place the automation application into the enterprise architecture. The application resides in the levels 1 and 2 in this division. That application is executed in a process controller, which is basically an industry grade pc. That in turn is connected to the level 0 by I/O interfaces. The automation application has multiple functions, it measures, sets, regulates, archives, registers, prompts and manipulates the values of the process [21].

Figure 2. *ISA-95 (International Society of Automation) reference model. Modified [22]*

The above levels can be further divided to application parts. Figure 3 displays the common information flow from one part to another in automation application. The figure is illustrative and not one to one mapping, as there exists more equipment than operators. The exact parts the automation application contains depend on the automation system and process.

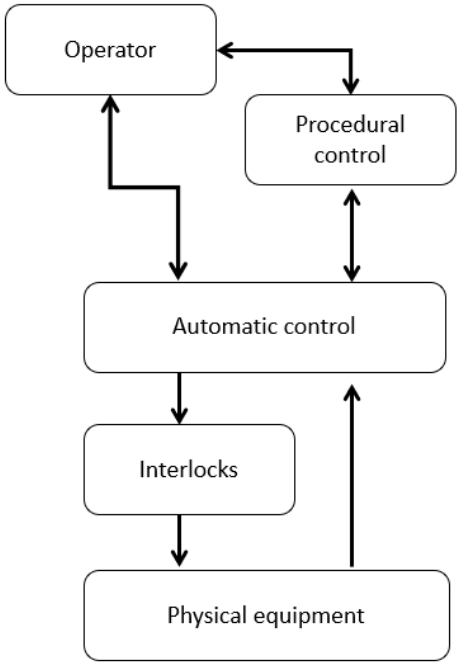

Figure 3. *Connections in the automation application. Modified [20]*

Operator instructs the automation application by monitoring and operating the automatic controls or by influencing the procedural controls of the application. Procedural controls in turn act with the automatic controls by receiving status information and triggering actions. The automatic control part's purpose is to maintain the steady process state, it contains the algorithms to adjust the process values properly. Before the physical device is controlled there are safety interlocks that prevent device or persons from harm.

Application format

Graphical notations have become a natural way in automation applications for historical reasons. Old systems used pneumatic and relay controls instead of computer-controlled systems and the relay connections could be visualised as ladder diagrams, this visualization has continued in the application domain also [13]. Electricians understand the program code that resembles them of the logical connections in electrical diagrams. Visualizing connections by wires in diagrams is a natural way for multidisciplinary field.

Visual diagrams are easy to comprehend until it becomes cluttered at some point. A diagram becomes hard to comprehend around 50 objects, this is known as "Deutsch limit" as quoted in [23]. Since it takes quite many of the basic function blocks to represent a single functionality, grouping of the functions is needed. In [15] Karaila writes that an overview of multiple connections hides complexity and makes understanding it easier. Standard structures should be used to note what has been subtracted on the abstract diagram.

The automation application is engineered using a vendor specific tooling. It is implemented using a graphical or text-based presentation language. The specific language depends on the vendor, the most common ones are the languages defined by IEC 61131-3 and often several of those provided.

The applications are built from a set of function blocks and functions in the selected automation system. In most platforms the function block level is static and closed. Those consist of basic blocks that are defined in IEC 61131-3 and vendor specific implementations for controlling devices such as valves. Function block development is independent of the project and done by software engineers using some higher-level language and model driven approaches [24].

3.1 Development of application

Application development, referred as application engineering or configuration is the activity which the predefined functions, function blocks, are strung together to perform a larger functionality [21]. When creating an application by configuring, connecting blocks together by wires the engineer does not need to consider the underlaying techniques such as reserving memory [25]. The application engineering consists of creating common elements for the design, implementing detail functionality, testing it and documenting it.

The application is engineered based on the design output from process engineering phase. Selection of the devices may be ongoing at the same time and so information may refine during the implementation. Functional requirements are the input data for the application engineer how the factory should work. Often used formats are briefly explained in the end of this chapter.

Application development starts by refining the functionality of function blocks by connecting and parametrizing to suit the need of multiple similar applications. This is done based on the device lists and by analysing the functional requirements for similarities. These applications are called templates or type circuits. Templates may be developed over time and usually contain years of expertise. The solutions can be either created by separate organisation or developed at need by a project organization and productized to support reuse. [12], [15], [24]

The main effort of the application development is the instance specific adaptation of the previously mentioned template. Application engineer takes a suitable template and par-

ametrizes it with the device information, they implement the safety interlockings and automatic controls by connecting application instances together, the exact logic is specified by the input data of application implementation phase.

The application can also be generated from a set of rules and input data. Such solution works well for repetitive functionality such as building automation [26]. The downside of the generative approach is ability to handle changes, either those need to be made to the generator, the input data or then the application should not be re-generated after any changes done manually.

Depending on the process the procedural controls are a large part of the application development. The procedural controls utilize the interface the automatic control part offers (figure 3). Those also vary substantially between projects [20]. These two aspects make the procedural controls less suitable for earlier mentioned template or generation approaches.

After the application design is finished the solutions created are tested against the specifications in several steps. First tests are carried out without the physical equipment then the equipment tests are executed without the process. Final testing is on site with the physical system. The automation application seldom is finished with one go from start to finish. It may take several iterations and changes might originate first to the specification and later to application or vice versa. This need for back and forth changes needs reverse engineering capabilities [27].

The tested application is documented as an “as built” document for the plant maintenance purposes. Documentation is time consuming and a specific document style may be required on certain markets to qualify as a supplier, for example [8] . Functional documentation is also almost totally repeating the same work that has been done when creating the application. Everything should be documented and in the same way it is implemented.

Input data

The input data for application development is provided in the format of spreadsheets, diagrams and textual descriptions. The contents and completeness of initial information vary between process areas, countries and projects. The information originates from the related disciplines, basic engineering, electrical engineering and process design. It is important to support multidisciplinary input as noted in [12].

Input data consists of device lists containing I/O assignations, device types, control limits or interlocking matrices that come in the format of spreadsheet. These can be processed

manually or usually by excel macros [28]. The application tools may provide a possibility of editing the data in a spreadsheet format, but the separate file acts a large role.

Piping and instrumentation diagrams (P&ID) are the main element of a design, those convey information about the plant floor devices and their interconnection to each other from the process perspective. These contain information of the control schemes for the application. The items in the diagrams are represented as symbols and a specific code system, tagging. [21] These diagrams often depict an area of the plant.

Along with P&ID there are control, interlocking, logic diagrams and flowcharts. These all specify the automation application functionality. Various visual appearances, combined functionalities and abstraction levels are present in such diagrams. In some cases the exact logic is copied by an automation engineer from a paper to the application. The format of the diagrams is heavily affected by what software those were created with.

Textual data may be used along or instead of the diagrams to specify the functionality. This is often plain text that tells either device by device or from a larger section that how that part should work.

Reuse

Automation projects often reuse some parts from older applications [29]. Reusing can happen on a device level if a similar device was successfully commissioned and tested elsewhere. It can happen also on control strategy level if a similar process part was controlled with the same principle. Even a full scope of the factory can be copied for a base to a new application implementation for similar plant.

Reusing requires pattern recognition from a similar implementation and the requirements. This is easier to achieve on a higher abstraction level. Current mechanisms of data transferring via hardcopies of drawings rely on the engineer's capability of pattern matching. In [20] it is said that the basic controls are already done by reusing existing application. They note that the effort goes into engineering the procedural controls which differ too much to be reused, so the reuse should instead be on a higher level where similarities exist.

Normal process automation application is far from a productized solution. The system it runs on is mostly productized. Love [21] notes that there are countless ways of controlling a process, some better than others. And the decision might come to the judgement of the application engineer. This decision would be diminished by reusing tested productized solutions.

3.2 Standards

ISA-106

This standard provides key models for automating manual procedures in continuous processes. Along with that it provides uniform terminology and modularization principles as well as best practises. It is created to ease the effort required to implement automatic controls of normally manual operations. The goal is to automate the process of starting or stopping the plant to ensure similar practises for efficient and safe operation. [9]

It provides three parallel models (figure 4) that drill down from business level to the device on the plant floor. It shares the same key models as ISA-88 for batch automation but for continuous automation. Models are physical, requirements and implementation. The physical model defines the devices on the factory. Requirements model describes the needs of control. The implementation model is the actual implementation based on the two first models. These models help to capture the required steps that can be automated by describing what must be done in each level. By defining the physical and requirement model it should be possible to implement the functionality. [9]

Requirement and physical models are structured into modules so the structure can be implemented also on the application. Each implementation module consists of the same aspects: executing commands, starting possible submodules and verifying that the result was achieved. Webinar [30] about the subject explains that the different levels would need to be independent regardless of the child or parent modules. And all hierarchies should always contain a control module.

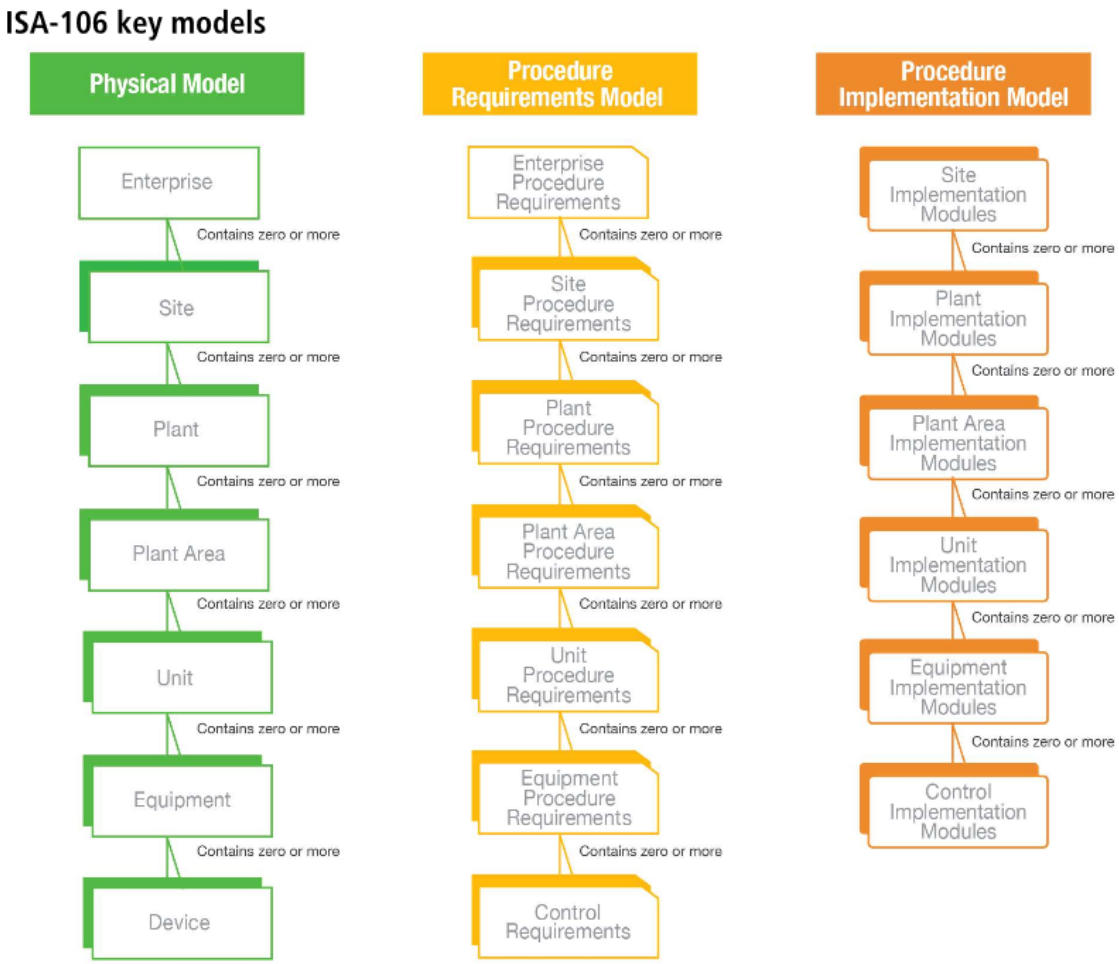

Figure 4. *ISA-106 three parallel models for describing and implementation [9]*

The hierarchical levels are used to form a process point of view. This is a view of a unit as a whole and not a collection of separate devices. Control applications implemented from this perspective allows for the operator to operate those with the same perspective. Another perspective that the standard assesses is state based control, in which the purpose is to organize the automated procedures based on the physical state of the unit. [31]

Yokogawa has a been involved in the standard committee and their offering contains Exapilot tool that aims to resolve the same problems as the standard. Exapilot is meant to be used on top of the existing control system to reduce manual operation needs. It offers typical procedural components to automate non routine work. These are produced in flow chart format with possible sub procedures. [32]

IEC 61131-3

IEC 61131-3 standard [33] is the most notable and followed in the field of automation systems. It is the third part of the IEC 61131 standard for programmable controllers and defines programming the industrial automation systems. It explains the basic concepts

and languages that are used in an automation system. Along with those it provides guide for structures, basic functions and elements of the building blocks.

It defines 5 languages for programming automation applications. Two textual and three graphical. Instruction list is an assembler like language that is to be deprecated from the standard. Structured text that is derived from pascal and is more expressive in complex operations than the graphical languages. Ladder diagram is analogous to electrical relay diagrams where contacts and coils are strung together between two rails. Function block diagram which represents functionality via blocks that represent program functions and wires. Sequential function chart provides sequential controlling of functions via steps that are either active or inactive and transitions between those.

Basic element of the automation in application in 61131-3 is a program organization unit. Those are either function, function block or program. These are used to modularize and structure the application. Functions are stateless algorithms that output the same state based on the inputs. Function blocks extend this and can contain their own state and memory. Programs are an assembly these two and supporting constructs such as global variable access. Execution of program organization unit is controlled by task, which provides periodic or triggered execution of associated program organisation units. Programs and function blocks can be instantiated in a resource, this is illustrated in figure 5. Resources are the processing units of the process controller.

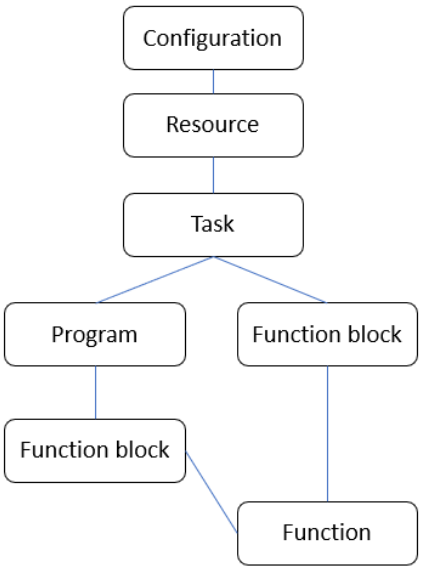

Figure 5. *Relation of configuration elements according to IEC-61131-3. Modified from [25]*

There are claims for and against the modeling capabilities of IEC 61131. [34] notes that IEC 61131-3 has some capabilities, just in a very low level. Function blocks can contain complex structures modularized by usage of other functions and function blocks in them. The restricting part is that interfaces to hardware are limited to programs and elements above it [10]. [20] notes that there is a wide gap in the semantics of the constructs in system models and the IEC 61131 offered models.

Contradicting with the claims above is that inheritance mechanisms could be used to model a hierarchical structure as in [1] is proposed. That seems to take advantage also of the capabilities of continuous function charts that are extended version of the defined function block language. The usage of partly specified directly represented variables could be a key point also for modularization. Also [35] suggests the hierarchical capabilities referring to nested function blocks as those can be built from other function blocks.

Vendor implementations can provide compliance fully or partly with the standard, some compliant systems are ABB Application Builder, Codesys, Honeywell ControlEdge and Siemens Step 7. Most of the systems allow for more functionality than the standard defines, such as continuous function charts that are more freely defined function block diagrams. The function block diagram is a language defined by IEC 61131-3 but is used as a broad term even for nonstandard diagrams.

IEC 61499-1

The standard [36] provides models for distributed control systems, concentrating on entity of function block. It focuses on event driven execution mechanisms of function blocks instead of familiar task-based execution as with programmable logic controllers mostly. This event-based execution should allow for more synchronized execution in a distributed system.

Each function block contains event and data interfaces. The event interfaces trigger the execution of the function block. This execution mechanism enables better synchronisation of distributed applications. It also requires less resources as the function blocks are not executing continuously. [37]

Function blocks contain an execution control chart, which consists of state machines. The various states are coded according to IEC 61131-3 languages, these states or algorithms are mapped to inputs of the function block so that in the arrival of an event an algorithm is executed. [38]

It allows decomposition in hierarchical direction with composite function bocks and by sub applications that are composite block like instances. Those can be distributed separately and can contain interface blocks allowing hardware access. It also includes an adapter model for connecting structured data between two instances, that decreases the visible interfaces and makes it easier and less prone to errors. These features enable modeling the software on a higher level. [39]

The standard continues to evolve as different interpretations are made of it. The writer argues in [40] that the standard is falsely appraised over IEC 61131 and that it does not solve the problems better. According to [41] designers are not aware and do not know how to utilize the options of the new standard. This may partly be caused by the fact that the major suppliers do not support the new standard or are just starting to support it.

Compliant programming tools are available from nxtStudio owned by Schneider Electric and ISaGRAF from Rockwell Automation. It is also possible to control other than their hardware with IEC 61499 with software that supports multiple hardware options such as 4DIAC or the nxtStudio.

3.3 Application tools

All the major distributed control system suppliers have their own application structures. Even if the application language would match it is possible to divide the application parts in different ways. The structure can be inbuilt to the toolset that is provided, or it can vary

between the use cases. As is noted in [12] the structures and ways of constructing automation application have a long history and new design paradigms have not gotten a hold on these systems. This is due to the nature of system critical applications and long development times.

Automation applications are made by large companies that do not boast with their application solutions in public so without working for them or being their customer, it is not possible to attain the solutions for closer view on application structure.

The following chapters analyse three vendors' current systems related to the subject of this thesis. Selected systems are Valmet DNA, Codesys and Siemens ecosystem. Analysis of Valmet's solution is largely based on work by Karaila [15] and writer's experience, Codesys is freely obtainable for a trial use and insights on Siemens are all from their documentation and workflow manuals.

3.3.1 Valmet

The automation application language in Valmet DNA is function block diagram that resembles the function block language defined in IEC 61131-3. The application consists of diagrams that are individual objects in the configuration database. Those objects contain one or more pages of function block diagram. The applications can be arranged into folders to create a structure. The applications refer to each other via global datapoint references.

The application is divided based on the devices, there exists an object as described above for each device on the plant floor. Two separate configuration objects exist for devices that are controlled from the system. These two objects act as a pair and are referred as child and parent objects. The child object contains the instance specific application logic. The parent object contains logic that is specific to the device type. This division is illustrated in figure 6. Other applications read the status of the device from the parent module and interconnections to the device are routed via the child module.

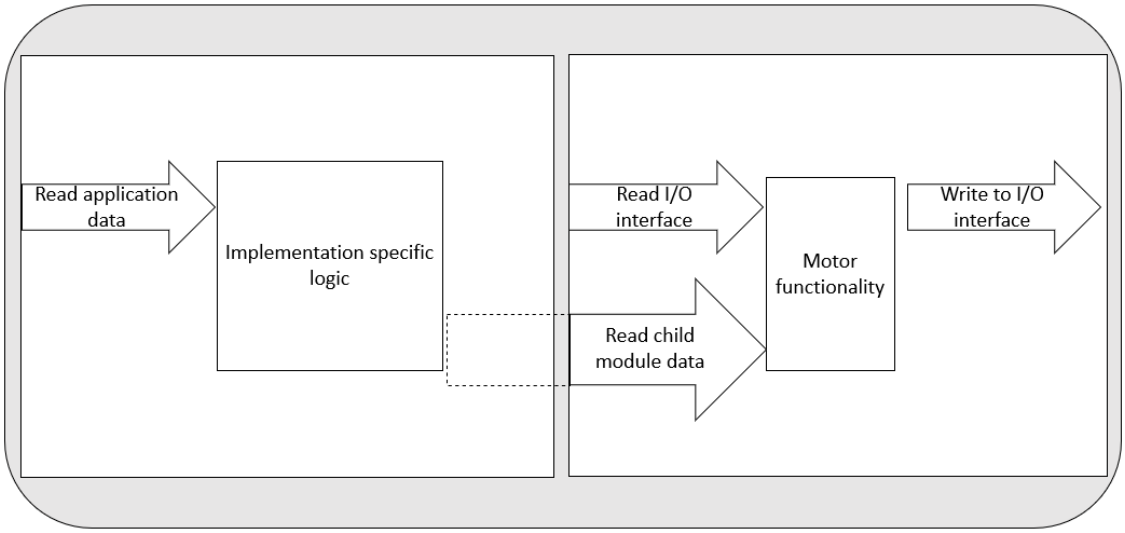

Figure 6. *Illustration of Valmet DNA application. The device logic is capsulated into a diagram on right that reads the control logic from its child module.*

Users can deviate from this practise and a function block diagram can be made for just about any purpose needed. That way the application can be structured into group control objects for example. The application has other hierarchy elements as in ISA-106 such as start-up sequences, but those contain only the algorithm part not any composed view of the underlying applications.

The applications have low abstraction level and features have been added to the function block diagram objects as the automation system has developed. That results as a cluttered view of the application which might be hard to understand.

Most of the applications are created with templating mechanism to reduce copying of the implementation and allowing parametrisation. The templating model is described in [15] and has not largely changed. Using templates allows for fast generation of applications that are error free but can still be modified case by case if needed.

3.3.2 Codesys

Codesys is not an automation system, it is a software platform on which one can create an automation application solution and run it on compatible hardware. Since this extends the possibilities for the application structure the following analysis is based on the object-oriented example explained in [42]. That example was selected due to the relevance on the thesis.

The object-oriented industrial programming example is marketed with punch line “Your plant is built from objects. Your control code should be too.”[42]. The example uses blocks to represent actual devices in continuous function chart programs. These objects

are nested into each other to make a larger entity. For example in figure 7, a motor is encapsulated in a conveyor motor, this with other encapsulated functionalities make up a reactor and that is instantiated in plant control containing multiple instances of the reactor.

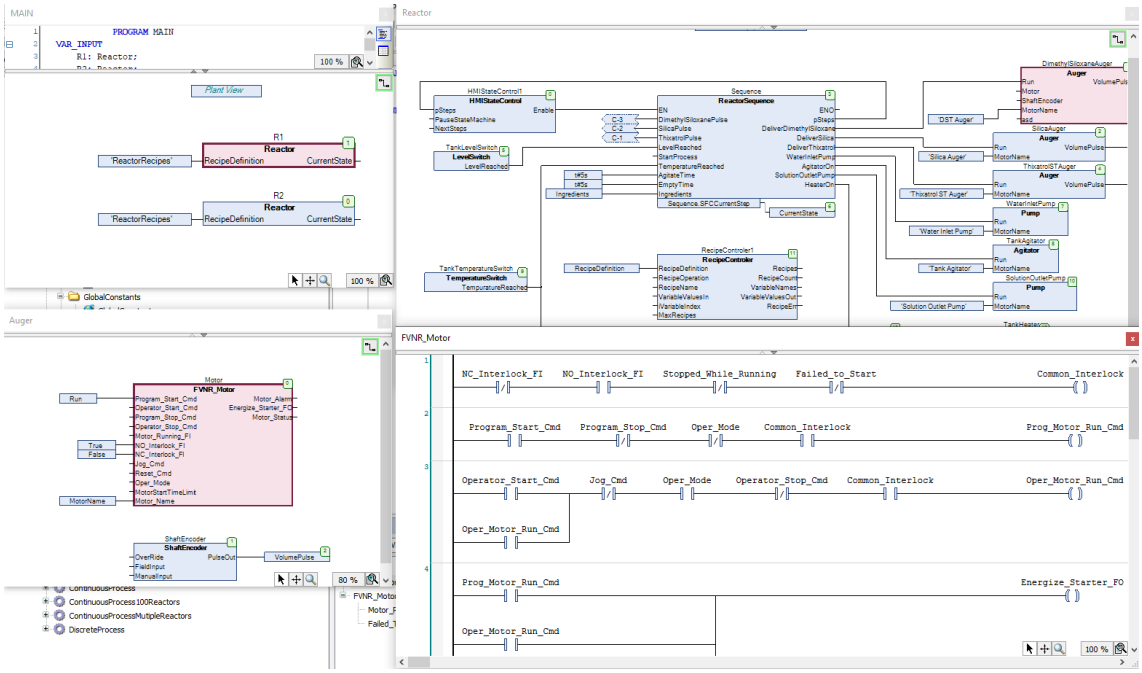

Figure 7. *Object oriented block diagram, in order of top left to bottom right, instantiated reactor, its implementation containing conveyor motor, the conveyor motor diagram and lastly its implementation.*

This nesting is beneficial in the use cases where several identical processes are instantiated with different parameters. Compared to the earlier Valmet's solution where similar application would consist of objects connected into series. This nesting is demonstrated to be powerful editing way of configuring tens of identical solutions. Instances are parametrized using configuration lists for example assigning I/O addresses.

Encapsulating the functionality works by combining functionalities and declaring them an interface. That creates abstraction layers, the user only needs to configure a level at a time and via set interfaces. That makes it easy to reuse the solutions. The solution does not offer any customisation on the individual devices other than trough parametrisation. There are no instance specific controls inside the objects. If compared to the ISA-106 the levels can be recognized along the way and procedural controls can affect the devices from outside connections that is shown as cross references in figure 8.

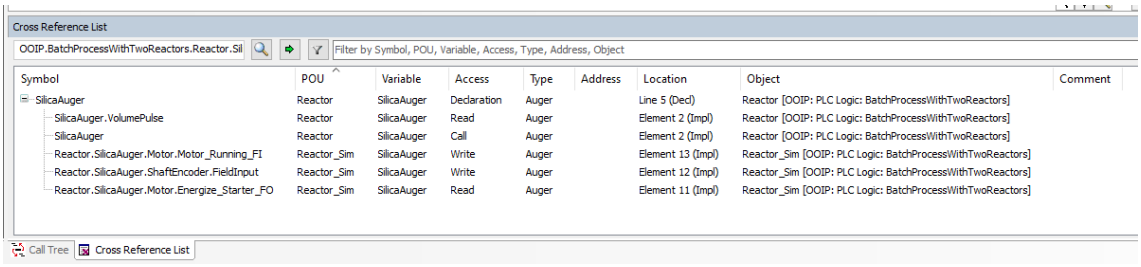

Figure 8. *List of cross references between the objects only seen on list.*

The instances are shown in the hierarchy as shown in figure 9. Only the top level is an actual instance (MAIN) and all the other objects are the ones that can be instantiated in the main application or in other objects. This has a closer relation to object-oriented programming as was the target of the example. It steers away from traditional process automation environment, as it has no individual control applications.

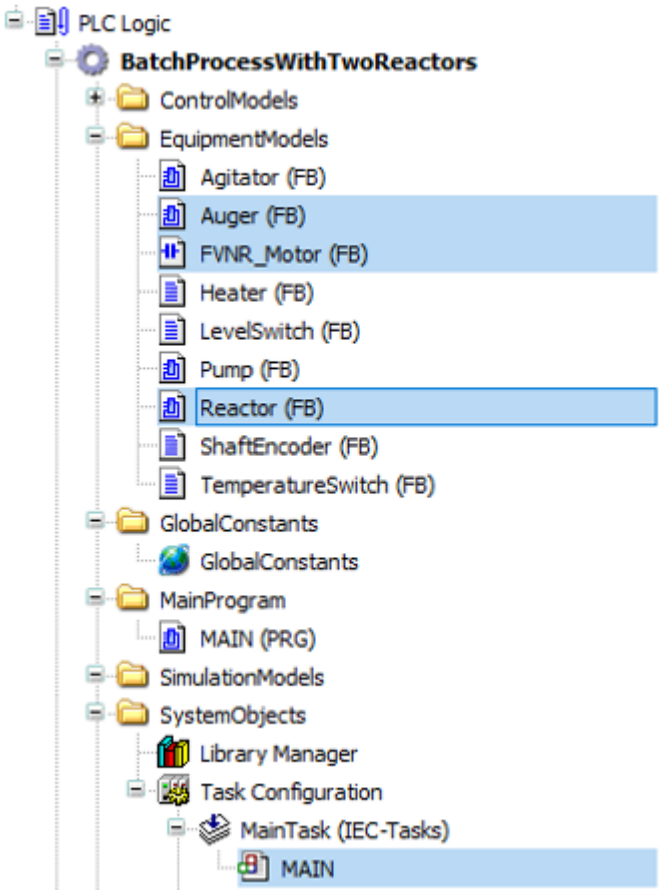

Figure 9. *Object tree view of the project visualizing the objects for the equipment and the main program.*

3.3.3 Siemens

Siemens is one of the most well-known automation suppliers. The following will focus their SIMATIC PCS 7 automation system. Analysis is based on the material available on Siemens web page as workflow and best practice manuals [43], [44]. Siemens portfolio

consists of several products that some can be used to achieve the same task. The focus is the abilities of plant automation accelerator and PCS 7.

The application structure is based on control modules that are continuous function charts. A control module contains a standardised control logic and provides an interface. It abstracts the view by making it possible to present a function block diagram as one block. Control modules are interconnected to each other and those are visualized in control module levels in figure 10. These modules are organized in a hierarchy view as shown in figure 11.

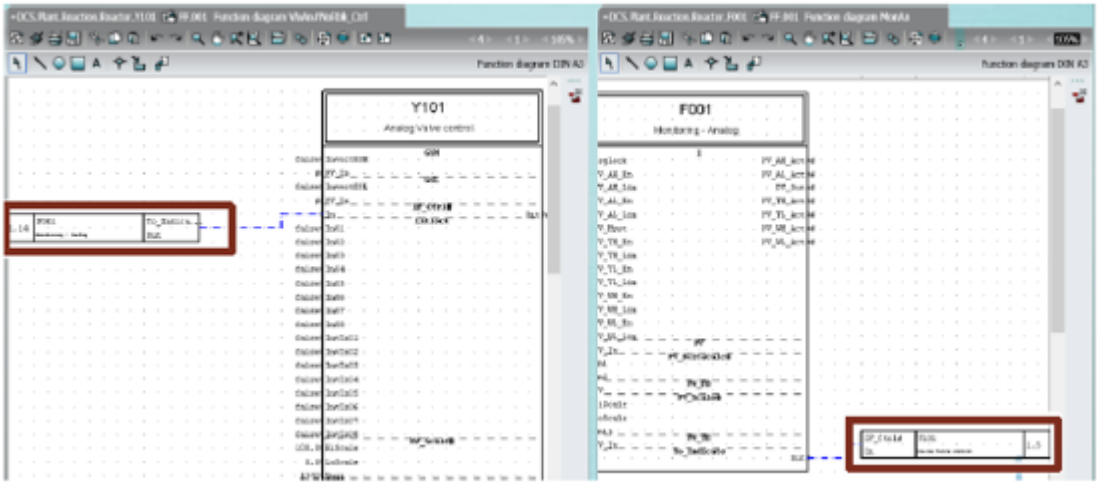

Figure 10. *Interconnections are visualized in control modules in both ends of the signal. The blocks represent function block diagrams and the dashed lines indicate separate functionality. [43]*

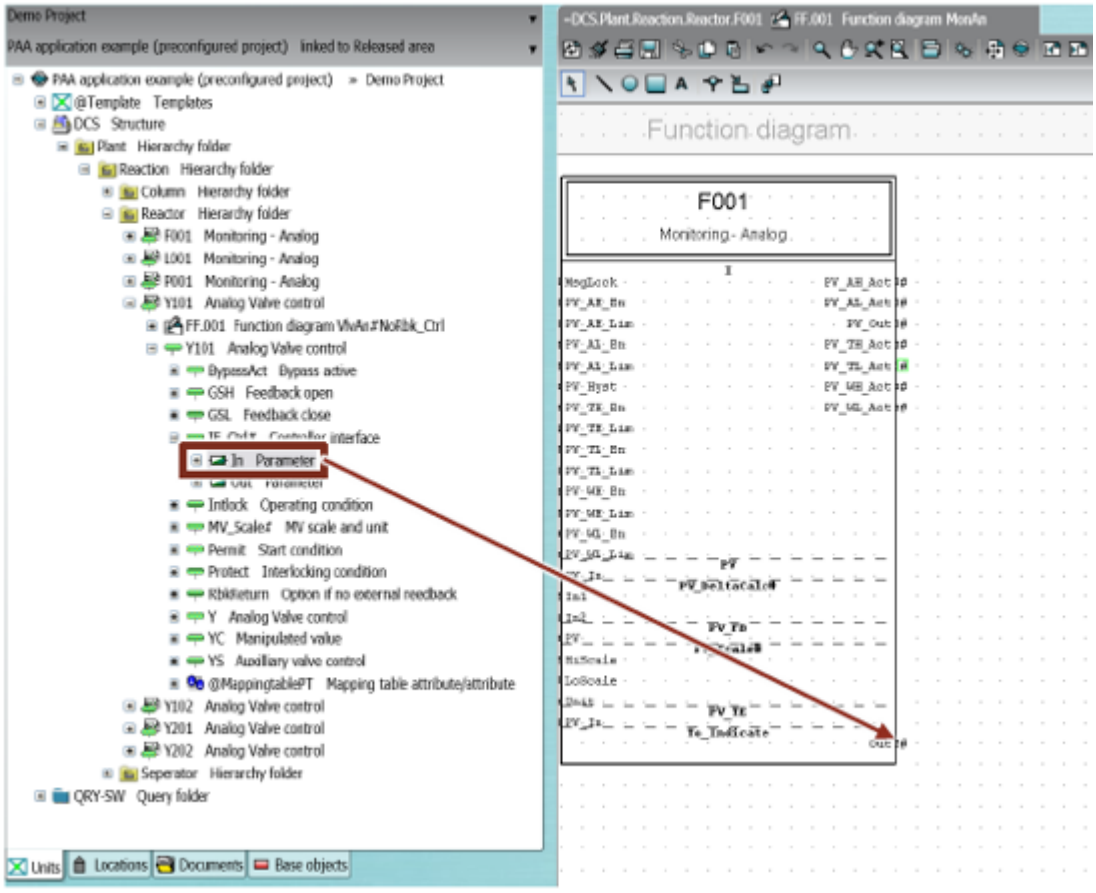

Figure 11. *A control module visualizes the functionality of a device in continuous function chart. Possible parameter and connection points are visualized in hierarchy and in the diagram. [43]*

Control module is an abstraction layer where the engineer sees and modifies all the interconnections and parameters of a single device but not the platform or device specific details. If compared to the two earlier solutions this level is like the Valmet DNA child modules but without the platform specific details. Or as a similar view as in Codesys example, but the underlying abstracted functionality is an instance so it can be edited separately.

The control modules are created from control module templates, the templating mechanism allows for fast implementation and change management as in Valmet DNA. Instances created from templates can be modified either by adding separate functionality to the control module without breaking the reference or by modifying the implementation via parameters.

Control modules that are a device specific entity are as far as the documentation tells the highest implementation level, there seems not to be a possibility for higher level of

abstraction. Solution templates that provide multiple control modules and interconnections exist. Control module provides similarity to individual control level and no higher levels are available in the application design.

Siemens Comos product provides the documentation style VGB to comply with [8]. It contains documentation options for higher abstraction levels and has some integration with the applications. The T-3000 product that is directed to power plants offers the compatibility option also. Where it is possible to select between displaying IEC or other symbols in the graphical representation. [45]

3.4 Other studies

Many projects on the area can be recognized, one being the Embedded systems Design Environment for the Industrial Automation sector (MEDEIA) [3] it is based on models in a hierarchical solution. They have introduced a middle component that specifies the objects communication and behaviour interfaces that makes hierarchical aggregation possible. The idea is to have the component between the specifications and the implementation. The component provides domain specific views that would be specification templates for the domain experts. Those can specify aspects or functionality in different ways. These specifications are stored in the automation component and can be transformed into different views or program code. [3], [27], [46], [47] The proposal looks promising but lacks critical view on feasibility. The amount of needed views for the described features would be large to maintain.

Similar concepts are proposed in AUKOTON process, which applies model-driven design in the format of UML profile. The process includes importing requirements into the model and refining it as platform independent model. The approach has been evaluated with industry experts with positive results. But the model was criticized that it did not match any familiar format the experts were used to. [28], [48], [49] The process looks promising with some development aspects. The authors concluded that the modeling look and terminology is not suitable for automation engineers, the same can be said about most of the academic proposals. Another point that was not clearly solved in the work was the unidirectional workflow. As presented in figure 12 the workflow is from left to right without possibilities to go back, and the code is generated at some point. That could mean that the maintenance engineers need to have the ability to generate code after changes and to understand this format of models.

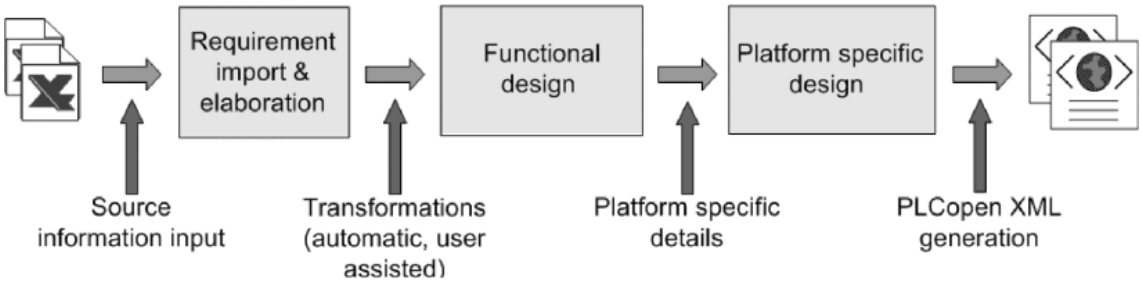

Figure 12. *Workflow of the AUKOTON process [48]*

The two previous works were mostly focused on the challenges of data transformations. Hierarchical solution is proposed in Monaco project [2] where a domain specific language is created for event-based programming of automation solutions. It does not try to cover

the whole automation application but the sequential control of machine. It supports hierarchical control which allows for abstraction of the lower components into uniform commands. A worthwhile aspect is a verification of the model against the functionality that the parts provide. Hierarchical model is proposed also in [4] where all hierarchy levels would be presented as functional basic components and implemented by either other functional basic components or at the lowest level any of the IEC 61131-3 defined languages.

The usability of several notations and methods have been analysed in several studies and following should be noted. Model-based engineering approaches should increase efficiency [50]. UML lacks the domain specific notations so SysML would be better [34]. Generation of the application from a model is possible but lacks the possibilities for bidirectional engineering [51]. New application languages create lack of capable engineers [52]. Legacy software needs to be supported [50]. Automated clone detection capability would be helpful to recognize the similarities and save in engineering costs [29].

4. RESEARCH METHODS

This thesis was done to find out what would be a suitable hierarchical format of an automation application that would enhance the productivity of automation application supplier. Some idea of hierarchical structure had been formed before this thesis. Qualitative interviews were selected as the main research method.

Background information on automation application is written partly based on the writer's experience from the field from last 5 years as well as literature sources. Other automation systems are explored in Chapter 3.3 as writers experience limited mainly to Valmet DNA. Literature review of studies on the field (Chapter 3.4) was included to get larger view on the subject. Based on these findings a small model was created to support user interviews that is documented in Chapter 5.

Qualitative interviewing is the key resource of obtaining information. It is conversation between peoples to obtain or to transfer knowledge of a certain subject [53].

Interviews are generally categorized into three structures: unstructured, semi-structured and structured. Which range from the most freeform to the strictest. The unstructured interview assumes that all the questions are not known to the interviewer and the person acts as a listener as the interviewee tells their life story [54].

Structured interviewee is on the other end of the spectrum. It consists of pre-defined questions scripted so that the interview follows the same pattern to produce organized and even quantifiable data [54]. This type of interview is close to a survey and it does not take the advantage of the dialogical possibilities in face to face situation [53].

There exist multiple points between the line from structured to unstructured, that are called semi-structured interviews, partly because of this variety it is the most popular selection for interview structure. It makes better use of the dialogue between interviewee and interviewer and gives the possibility to follow certain angle. [53]

The number of interviewees on one session influences the outcome. Group interview or a focus-group, is a type of interview where the interviewer acts as a moderator in a group conversation having more than one interviewee. This brings out the interviewee's attitudes and opinions with its flexible discussion. These are suitable for exploratory studies in unknown domains. [53]

Individual interviews where there is only one interviewee might be less colourful. These are easier for the interviewer to guide. It is also possible that in a group conversation

some opinions are not heard [53]. It is also easier to find suitable time for the interviews if there is only one interviewee.

Several other aspects affect the outcome of an interview, some to note are style of the questions or the atmosphere [53]. The perspective of the interview is also important. The perspective can either be to record the sayings of the interviewees, or to create knowledge based on the event itself [54].

Interviewees play a large role in the outcome, who should be interviewed and how many interviews should there be. Selection of interviewees comes to selecting the target group who are focused and sampling the individuals of that group. Sampling can happen randomly to prevent biased opinions or by selecting individuals with a goal in mind such as maximum variance. This may also be limited by the willing participants or given resources. There seems to be no correct number of interviewees, only one is enough if that person is the only one who knows about the subject. Too many on the other hand make the data amount massive and the interviewees opinions less transparent. [53]

5. MODEL

To support the interviews model of a sub process of pulp preparation was created. Refiner was selected as it is an independent unit of reasonable size. The selected example consists of 4 motors, 5 valves, 9 measurements and one controller. Model of this size is not capable of producing relevant information to higher level abstractions. It should be enough to provide some information and idea on the subject.

It is surprisingly hard to get hang of the functionality of the process by the initial information provided which came in format of individual logic drawings. Those diagrams contained control logic individually for each device. First the whole application was drawn as a single diagram that displays all the connections of the devices, that quickly showed that even though in Chapter 3 it was mentioned the limit of function blocks on a canvas to be 50 it was not easy to understand functionality when the amount exceeded 10 when there were multiple interconnections between those.

Application experts are not programmers and modeling languages such as UML look strange for them. They are however familiar with notations such as function block diagram. That can be used to model the application to a certain level of abstraction. In this model the visual aspects of the function block language were used. Function block diagrams were abstracted and displayed as function blocks in other diagrams. That way data transfer between application modules is displayed as connections between function blocks. That ends up being like what has been suggested in [15] and [4].

The proposed model uses the same division as in current Valmet DNA applications with the child and parent objects but turning the relationship. The differing part comes from being able to build visual connections between the applications and design controls for group of applications the same way as for single device. A valve is used as an example to present this solution in figure 13.

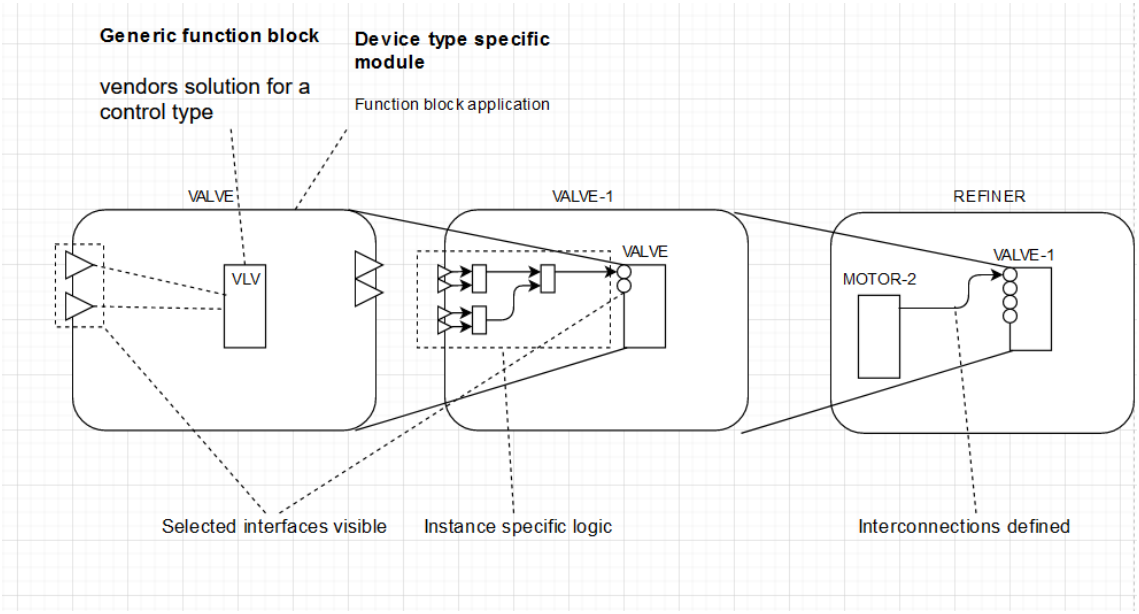

Figure 13. *Illustration of hierarchical composition of different levels in automation application.*

On the left side of the figure 13 a generic valve function block is used in a typical valve application with parameters to handle field signals. This application has an abstracted interface so it can be used in individual level where the implementation specific interlocking and control signals can be defined. In the last level on the right these connections are visualised in a diagram that groups the whole functionality together. Idea behind that is that one could define the functionality from right to left. Starting by adding devices in a group and defining relationships between the components, then progressing to more refine the logic in the middle level and lastly specifying the exact valve and its and hardware addresses.

The full view of the interconnected devices resulted in 5 valves, 4 motors, 1 controller and 8 measurements as the application in DNA. The group level requires 5 status signals from outside the group and a controlling sequence that was not included into the group. Inside the group 22 unique signals are connected totalling of 82 point to point connections. In figure 14 the whole refiner application is shown but half of the connections are hidden since in some cases multiple variables were transferred between applications, and those are visualized as one connection. The external signals are marked as ext1-5 and sequence controls just as a sequence connection. The connection specifiers, that indicate the function of the connection have been omitted due to lack of space in picture format.

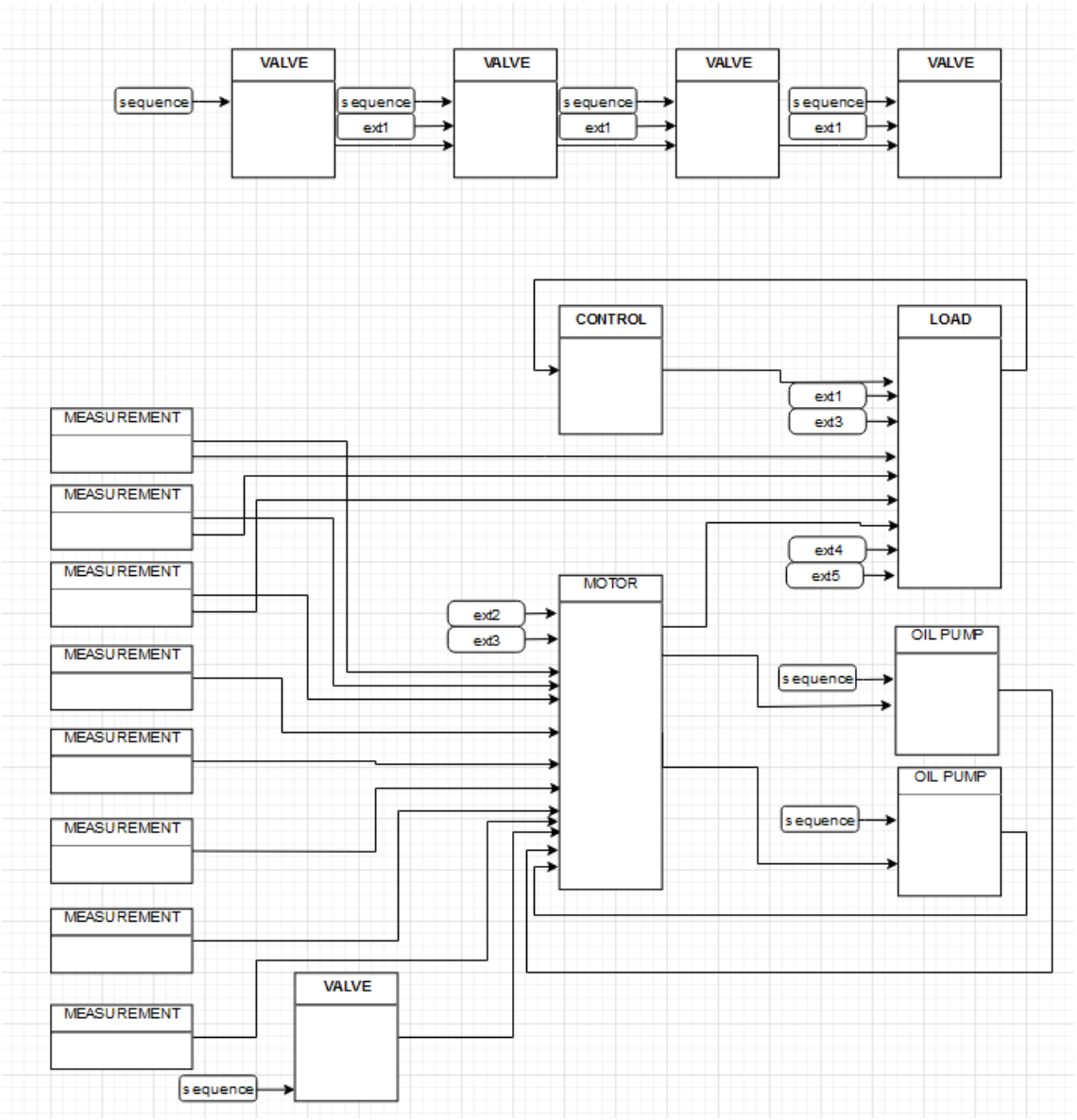

Figure 14. *Refiner application, exact names and datatypes are left out from the picture.*

6. INTERVIEWS

Interviews contained themes: application structure, application engineering, levels and tasks of application and the visual aspect of the application. The main questions to structure the interviews are listed in table 1. Follow up questions are formed during the conversations depending on the answers. Interview structure contained an introduction to hierarchical engineering and the model presented in Chapter 5.

Table 1. *Main questions of the interviews*

Topic	**Questions**
Application structure	What kind of structure should an automation application have? How do you see the Valmet DNA application structure?
Engineering	How would you like to design an automation application? How is application currently designed? What format is the initial information provided and what affects to it? Why is the same design process repeated in several phases?
Levels and tasks	What levels should the application have? What tasks are needed in automation application?
Visualisation	What is the proper visualisation style of automation application? What information do the engineers need to see of the application?

Interviews were selected as a research method to get data of current automation application engineering practices and the effecting variables. Idea was also to collect development ideas from experienced engineer.

Interviews were conducted as face to face interviews for most beneficial results since the subject might not be familiar to all interviewees, conversation would be more natural

in that situation. For that reason, semi-structured interview structure was selected. It allows more probing questions to follow up on the subject interviewees might have knowledge about but keeps the focus on the things that were under research.

Interviewees were selected based on their experience on automation field. Preference would have been to find persons who had worked with different systems, but the persons were limited to those working at the same location and employer. A limiting factor was also the availability of the interviewees at the time of the interviews. Selection was done from several groups so that the persons interviewed represented groups working with engineering and lead engineering activities, development, product management as well as sales. These groups were selected as they all bring a different view on the subject.

Interviews were conducted as individual interviews in the autumn and winter of 2019-2020 in Tampere by the undersigned and a usability expert. Interviews were around 1.5h long each and recorder in writing. In total there were 5 interviewees.

Interview topics

The focus of the interview should be on two points, what would be the application structure and how would they like to design the application. It should be clear to the interviewees that the change in the application structure would be to ease up their work as engineering would be easier and faster. The interviews are about studying new ways of engineering the application so there is no need to think about the current limitations of the automation system. Different aspects should be prioritized based on how often those would occur in the engineering phase and how much time do those consume.

Application structure

In this section the interviewees should be given their own say before presenting the example model to prevent guiding their answers. This should bring out differing opinions on how the application could be structured but the years of engineering with Valmet DNA can result in no development ideas. For example, they might remember times before the Valmet DNA got the application structure it now has and still praise that change.

The design principles and connection methods between applications are a key aspect of the structure. Interviewees' opinions about changing from device centric control applications into more abstract group level controls is worth noting. Current understanding is based on the previous chapters that engineers need to understand every part of the application and that the details visible in Valmet DNA application are important.

Valmet DNA's current application structure has been successfully used for around fifteen years and has been thoroughly tested. The best aspects of that format should be also discussed. It would be also interesting to hear about the usage of library solutions.

Engineering

Something that did not come up clearly on the analysis in Chapter 3 was the way that the application should be engineered. Most of those only focused the format of the ready application, not how application is engineered. The change to top down engineering method should be gone through with the interviewees.

The idea of redoing every part of the application engineering in terms of specification, application and documentation is not productive. Do the interviewees see it also as duplicate work or are there important parts in the steps?

The structure of the input data is also an area of discussion. As mentioned in Chapter 3.1, it differs a lot depending on the project. Initial thought has been that some parts of the specification workflow could be left out if the application model and engineering would flex to fit the usage.

Division into levels and tasks

Dividing applications into hierarchical levels with different abstraction levels is the generic idea of the thesis. Some levels are presented in Chapters 2.2 and 3.2. What levels would the users want to have and what are the different uses? It seems that there are multiple options, and some may be limited to certain process section

The application is also categorized by purpose [8] , that guideline is the only source in which the division is so detailed. The interviewees opinions should give insight on the need of dividing the application on separate tasks.

Visualization of application

Different visualization styles could be useful for different user groups or to differentiate the application levels from each other. There could be visualisation styles for different application levels in the application tools or in configuration database.

Key part of engineering applications on a more abstract level would be to leave some parts out. Should this happen automatically and what is the engineer's role in the operation? Is something like that already done on initial information that is provided?

The connections between the applications have several development ideas to be discussed. The VGB guideline [8] suggests using different connection style between hierarchy levels. The hierarchy levels would change the connections that currently refer to

other devices control applications. And is it possible to combine connections for a clearer view?

7. RESULTS

This chapter evaluates the format of the hierarchical model of automation application. The results on this section are based on the interview notes. These are assessed against the findings of other studies and industry practises presented in Chapter 3. The chapter is structured following the interviews presented in Chapter 6 and summarized in the discussion section.

Many of the interviewees had much to say about the problems of the current system but not so much about the development potential and that left some of the questions unanswered. It was also affected by the time reserved, even some interviews took another session totalling in over 2 hours. Interviewees criticized if the examples were not relevant to their field of expertise and that those too simple as examples. They also noted that the VGB guideline [8] that was referred in the interviews is valid mostly for Central Europe power industry and not the whole automation field.

7.1 Application structure

The interviewees were asked about the desired application structure that they would like to work with before introducing the model presented in Chapter 5. Some of the interviewees had no ideas about what to answer to such question, they had worked with the given limitations for so long. Most provided answers that how have they coped with the structure of Valmet DNA automation system.

On general level they agreed that the desired application structure differs on the perspective and phase of project. For the initial specification and the final operator, the application is viewed more as one entity, something goes in and the process happens. For the device testing and start-up phase the single devices that make up the process are more important. For the engineers designing the application it comes to the fact that what is the easiest way from the requirements to the functionality, if requirements are not structured then creating a structure out of those is extra work. Their view was that currently the initial information is not structured as hierarchical functionalities.

7.1.1 Modular

Modularity was a key word the interviewees used. That was expected as other studies referred to modularity needs from users also [50]. Users wanted to have the application

split into separate pieces. The content of these pieces was different regarding the background of the interviewee. The engineers wanted to split the smaller functionalities, the device level controls into separate units. Sales and product management on the other hand wanted to have larger modular components that consist of the devices. One should be able to buy and commission a software module as it would be a hardware module.

The need for dividing existing levels into even smaller pieces comes mainly from what was noted in 3.3.1 Chapter that the Valmet DNA structure does not limit what functionality is inserted in the diagrams. That creates an application that is hard to grasp from outside. The structure should reflect the functionality without opening the diagram of the application. There should also exist a clear guideline that where should extra functionality be implemented if needed.

Application done with modules that are visualized in higher hierarchy levels would be easier to understand. On the other hand, some interviewees saw the need of displaying detailed controls. The format of the initial information also drives towards this direction, the modules the engineers would like to see on the application are not present on the requirements.

7.1.2 Hierarchy

Conversations provided multiple options for the use of the visual hierarchies. Those options were:

There is no need of a hierarchical solution. An interviewee thought that the application is just single devices and interconnections, there is no higher controlling structure except for cascade controllers.

Displaying the interconnections between applications. Meaning that applications exist as separate objects but are grouped and a visual appearance is generated to indicate the interconnected application. This comes quite close to what was proposed in the example model in Chapter 5. This is the simplest but least beneficial approach. It would also require less changes to the format of the initial information.

It is a place for common logic. It would clearly indicate that something has effect on a larger area than just a single device. Some interviewees had done some generic modules containing common logic.

The hierarchy would represent the interface and functionality of that level and contain the connections of its child modules. This was the preferred solution by the interviewees. They had little ideas or details on how to achieve such structure.

Interviewees opinions did not contradict with what was learnt in Chapters 2 and 3 but the majority would not fully support the strictly hierarchical model that was proposed for example in [2] and [4]. Since it would limit the user, they would allow the application to be hierarchical but keep the existing way of not using the hierarchy levels.

7.1.3 Interfaces

In Valmet DNA it is not clear for engineer that what interface does a device control offer. The interfaces may change from project to another.

The model would allow creating interfaces for each hierarchical level instead of forming the status in multiple control applications separately. These interfaces can be utilized to verify process states. The interfaces would also work for commanding process sections instead of the single devices. The current DNA model depends on the engineer that has created it. It might contain a common control application as described above, or all the devices are started individually. Another option used is that the applications follow a single device's state.

A visual interface would improve the usability as engineers need not remember the connection names, users could adapt to using the system more easily. Interviewees thought that the application engineering or configuration should be more like plugging a cable instead of referring to named memory addresses.

7.1.4 Valmet DNA

Some interviewees thought that the engineering with Valmet DNA is as efficient as it can be with the application structure it has. They noted that application can be edited without visually viewing and it brings more flexibility. But the tools and mass editing possibilities work on principle of text editing, by finding and replacing strings without context knowledge.

Interviewees agreed that the applications are too cluttered and reveal too much implementation that should be internal. General view was that those should be hidden, and the application engineer should only be responsible of the project specific functionalities. Some of the clutter in Valmet DNA side comes from the need to respond to customer requests that is done by adding features on top of existing.

As previously mentioned in Chapter 3.1, different reuse practices exist, reusing old applications increases the productivity. The interviewees had used old projects as examples and as a base for a new project. None had used the library templates straight without modifications and found that the reuse library was not suitable to find implementation.

The parts they had used had come mostly from their own old projects and those might have been compatible with the library a decade ago.

Reason why library solutions were not used was that those lack the features that would be used in the project. Those being the features that they had built on their earlier projects. Reusing solutions from the reuse library was also not seen as option. Reason that they gave was the amount of the applications, there is too many applications and searching those is not effective.

All the interviewed engineers appreciated the format of 2 separate modules for division of the project specific implementation and the device specific part. An interviewee told that the Valmet DNA way of dividing the application into 2 parts, has a history from a case where the implementation and preparation for testing needed to be started before any requirements came. Information about the device types was known. So, application was made divided to device specific part that could be already tested and a separate part where the project functionality would be then added along the project. Another factor was the ability to change the individual logic without disturbing the device functionality.

7.2 Engineering the application

The interviewees saw that the hierarchical format would bring alternative ways for the configuration of the application. The separate levels were seen useful for different use cases and maybe for different persons depending on their role on the project.

Some of the interviewees proposed that the applications and hierarchies would contain all possible functionalities readymade, even if it is not needed. This way the application could easily be expanded with the functionality by selecting it on the hierarchy and there would be no need to search it from a library or create it by them self. Of course, this would add a lot of unnecessary application.

They saw that engineering tools would need to support the format that the initial information comes, since it cannot be influenced in most cases. Reasons for not being able to change the initial information format were said to be the tools that the companies providing the initial information used. For retrofit cases, the old functionality is reproduced based on the old application.

Interviewees had also noticed the claim that some things are made multiple times in the workflow from functional specification to automation application and into documenting the functionality. They had not needed to provide the documentation in the format defined in [8] but saw it beneficial if it would require no extra steps.

Especially helpful the hierarchical format was seen for connecting devices together to prevent typing errors in the prefix or postfix of the applications. This had not come up before the interviews to be a problem. That problem is in Valmet DNA varying convention of signal names that can change from project to project.

Some of the interviewees had doubts about engineering with larger structures since current practices of starting application engineering rely on generating the device specific applications. It was hard to promote any other approach as they are getting most of the applications from list-based tools by generating the application currently.

Hierarchical levels should help connecting the correct devices together but forcing the application into a strict format will affect the outcome negatively. The solutions where hierarchies could be best utilized would be the productized solutions where the application structure is designed from the start. The format also helps testing the solution, mostly if the format matches the specification.

7.3 Levels and tasks

In Valmet DNA the application hierarchy, just a folder structure, depends heavily on the project. It might not resemble the plant correctly or it might be just folders created to divide devices by their type. This is caused partly because of the differing practises between engineers and products. They thought that the applications should reside in the level that the application affects, and in the correct location. Even some forcing of application location could be enforced to prevent placing applications in wrong hierarchies.

It was proposed to divide the application always to four levels. The bottom level would contain device specific applications, the second level any low-level sequences that are used in process areas, third level the start-up sequence and production balancing or pollution controlling. The final topmost level would contain reporting features related to performance of the plant.

The users pointed that the individual control level is always needed and needs to be modifiable, even that it belongs to a larger group. This confirmed what was suspected in Chapter 3.3.2 in the Codesys example, that only generic inherited modules cannot be used. There was also a comment that in some projects there is no common control nor need for any hierarchy levels. If this is the case, then there should be a possibility to engineer the application without coming up with separate levels for it.

Some interviewees recognized the capabilities of using the hierarchy levels as parts of the designation system. The identifier would flow from top down and on each level a unique identifier would be appended to the identification code.

The separate tasks did not get much support from the interviewees, division to different functionalities seemed artificial and to break the functionality into separate pieces. They did mention that seeing the separate tasks at one time might help but the overall picture would be more important. Tasks measuring, actuating, monitoring and automatic control mentioned in [8] were not seen as different that would need separation at all.

7.4 Visualisation

Application diagrams

Engineers that are used to visualizing automation application as function blocks are not keen on adapting or even requiring symbolic visualization like on the user interface. Meaning that components need not look like the drawing symbols on P&I diagrams in engineering phase or in the application. That is a contradictory result to what proposed in [13].

The function block application should visually represent what it does, not that how it is internally built using the options that the automation system provides. Valmet DNA contains lot of application that the user does not need in most cases. One should be able to use the hierarchy levels for zooming in or out on the implementation detail.

The specific type of the devices should not affect the application design. A placeholder block was proposed for the cases where the exact device is not known. It should contain generic interfaces so that the application can be designed, and the placeholder can be replaced later. The idea seems to rise since the interfaces in existing applications are not uniform.

Interviewees had a common view that the automation application should be represented as function block diagram and use the same mechanisms. They saw it useful that it would indicate the hierarchy level that the diagram represents even if it could be deducted of the devices or structures that the blocks represent.

Interfaces

Visualizing the application interface caused concerns about the size of the interfaces that might consist up to 100 separate members. Displaying those all in any meaningful format would be problematic and not clear in any way. Users should be able to select the displayed interfaces from the available interfaces. The interviewees were used to many connections and possibilities. But having that many possibilities presents a steeper learning curve. There are limitations to hiding connections if it is necessary to display every connection in the basic control level.

Some interviewees supported categorizing the interfaces the same way as in figure 10 and in [4], either by their type or purpose. Application could have different types of interfaces that an application regardless of level would either support or not. Some suggested making templates out of the interface groups that multiple signals could be presented as one, as was done in the example in Chapter 5. The interviewees did not agree on whether there exist so similar connections that those can be templated or not.

Connections

There was no unanimity on how the connection or addresses between the applications should be displayed in the basic control levels if those connections go through higher levels. The proposed levels would change the signal target as the data goes through another application. The signal target names are important for understanding the application functionality.

Options for hierarchical connections were that the low level would inherit the reference from the upper level if possible, if not possible the reference would reflect the location or address of the upper level. Another option would be that the lower levels would not contain that information, just that it would provide an interface without any reference. The first one got more support in terms of readability and debugging for it requires less opening the applications, the latter one would not bring enough information in printouts for example where navigational features are non-existent. This division depends if the application logic is moved to other levels or not. If more of the application logic were done in the upper levels, then there would be less needs to display the references inside the application. In the Codesys example in Chapter 3.3.2, a single level is unaware of the connection targets.

The best option that could be deducted from the interviews is to display the references as [8] VGB guideline suggest. Referring to the functional unit the signal originates from. The reference should contain the location of the incoming signal and the area it originates from.

The Valmet DNA way of displaying the references was noticeable from the answers of the interviewees. They supported the way that Valmet DNA writes or reads data and the other side of that connection does not visualize the connection the same way, but a different kind reference is required there also. This differs from the way the initial information is provided as in those the signals are marked mostly on both ends and as in Chapter 3.3.3 Siemens displays it. They were not straight against the always present connectors but thought that it might make the diagram more cluttered. Requirements

from the interviewees were visual que about the connection and possibility to navigate to the connection target.

7.5 Discussion

The findings above proved that hierarchical application structure would benefit some phases in application engineering and make larger application solutions easier to understand and use. It does not fit into every use case, and with current practises would require larger initial effort. The results summarised below should be viewed with criticism since many of the points made were biased by the background of the interviewees.

The application should be built as a function block diagram. It should contain optional hierarchy levels, that are used to design the application using larger structures. The visual design elements in such diagrams should represent other control applications. These levels should provide the user a view, that helps them achieve the task they are working with.

These design elements should have uniform interfaces and be independent modules that can be configured together. If some modules are optional features in a product, there should exist a way to disable or enable those as is done in Valmet and Siemens templates. Regardless of the ready features or the application levels, all parts of the design should be modifiable or replaceable.

Only one hierarchy level should be a requirement, the device level. Along with that the individual control level, like the child module in Chapter 3.3.1 in Valmet DNA, should exist for controlled devices. Other levels build on top of these two, it should be guided what those contain but not restricted.

Application engineering needs to be possible by starting with the larger structures and progressing down and by creating the device level first and moving up. The diagram appearance should be as clear as possible on the higher levels to use the same tools for specifying the functionality. Productized models are the first option in which the hierarchies should be utilized.

Findings from the interviews were similar that were pointed in Chapter 3.4. As in [28] was found out generating automation application based on initial information is preferred. To fully utilize the benefits of hierarchical application the workflow would need to change from generating the applications into more of parametrizing the generic structures with the data.

The main “problem” is that the functional requirements are decomposed down to the corresponding physical devices. The input data that the application engineer receives is this decomposed information and engineering the application is easiest in the format that it is structured. A requirement model such as ISA-106 suggests in figure 4 would help that. The application could be defined and implemented with the same levels of hierarchy. The control structures are more generic on more abstract requirements before knowing the specific devices, making the controls reusable and understandable.

None of the interviewees referred to model-based engineering as a term or even to the modeling languages presented in Chapter 2.4. That does not mean that modeling would not be suitable, that just the engineers do not care what it is called. The diagrams could be models and the abstract views achieved by model transformations. Even the visual appearance and semantics for familiar experience should be achievable trough a UML profile.

Comparing the interview results against the automation systems in Chapter 3.3 shows that most answers were to improve the current Valmet DNA application. Some improvement ideas that the interviewees made could be seen from the Siemens examples already. Referencing more systems could have brought up more hierarchical application structures.

8. CONCLUSION

Automation application design process follows mostly the same patterns. Process engineer designs the process functionality and records it in a diagram that follows the format that they are used to. The automation engineer translates that functionality into automation application that follows the restrictions the automation system sets for it. And then application is documented as it was implemented. In this thesis a hierarchical model building up form the device specific applications is studied.

The thesis has gathered views on the structure and design of an automation application in relation to the hierarchical structure of the application in continuous process automation. The starting point was to improve the design efficiency by changing the structure of the application from individual device-specific controls to managing structures. The main method of the study are individual user interviews.

The findings suggest that a modular application built from hierarchies that represent units in the process has benefits. These units should provide proper interfaces and visualise the devices but to still maintain the editing possibilities of each level separately. The application format should not force the usage of hierarchy levels and it should be built with function block language to preserve efficient workflow.

The proposed hierarchical application structure makes handling and understanding of larger entities better. Efficiency of application engineering could not be verified, the largest benefits would probably be achieved from usage of products that are built hierarchically. If the input data for application engineering from basic design does not change the hierarchical solution causes extra effort for the engineers and may not be feasible.

The topic is broad and requires further development work. In order for the idea to be fully functional other areas than just application structure should change. The automation application should be considered in the systems functional decomposition phase, and functionality should not be decomposed fully into the device level. Further studies should consider changing the input data of application engineering and including user interface as a part of the study. Any proof of concepts on the subject should be designed from start with a hierarchical structure in mind.

REFERENCES

[1] Gary Pratt, Standardizing control system programming with IEC 61131-3, Control Engineering, CFE Media, Mar 1, 2017, pp. 44-47.

[2] H. Prähofer *et al*, Monaco—A domain-specific language solution for reactive process control programming with hierarchical components, Computer Languages, Systems & Structures, 2013, pp. 67-94.

[3] T. Strasser *et al*, Multi-domain model-driven design of industrial automation and control systems, IEEE, 2008, Conference proceeding, 2008 IEEE International Conference on Emerging Technologies and Factory Automation.

[4] N. Iriondo *et al*, A methodology for hierarchical industrial control systems: application to a heat treatment line, UKAC-IEE Control, 2004.

[5] INCOSE, Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, Wiley-Blackwell, 2015.

[6] Stanko Strmčnik, Đani Juričić, Janko Petrovčič, Vladimir Jovan, Theory versus practice, Case Studies in Control : Putting Theory to Work, Springer, London, 2013, pp. 1-37.

[7] D. M. Buede and W. D. Miller, The Engineering Design of Systems : Models and Methods, Wiley, Hoboken, New Jersey, 2016.

[8] Function Related Documentation of Power Plant Instrumentation and Control in Line with Operating Requirements, VGB PowerTech e.V, Essen, 2004.

[9] M. Wilkins and M. Tennant, ISA-106 and concepts of procedural automation, INTECH, 2015, pp. 34.

[10] Teollisuuden järjestelmät, asennukset ja laitteet sekä teollisuustuotteet. Jäsentelyn periaatteet ja viitetunnukset. Osa 1: Perussäännöt, Translation of EN 81346-1, SFS, 2010.

[11] Tekninen Dokumentointi: Viitetunnusjärjestelmä ja sovellutukset, Suomen standardisoimisliitto, Helsinki, 2011.

[12] D. Hästbacka, Developing modern industrial control applications: on information models, methods and processes for distributed engineering, Tampere University of Technology, 2013.

[13] U. Katzke and B. Vogel-Heuser, Combining UML with IEC 61131-3 languages to preserve the usability of graphical notations in the software development of complex automation systems, 2007, Conference proceeding, IFAC Proceedings Volumes (IFAC-PapersOnline).

[14] C. Wagner *et al*, Requirements for the next generation automation solution of rolling mills, IEEE, 2017, Conference proceeding, IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society.

[15] M. Karaila, Domain-specific template-based visual language and tools for automation industry, Tampereen teknillinen yliopisto, 2010.

[16] H. Rudele and F. Kantz, Improving program flexibility and application engineering for decentralized control in factory automation, IEEE, 2008, pp. 66-69, Conference proceeding, 2008 IEEE International Conference on Emerging Technologies and Factory Automation.

[17] S. Beydeda, M. Book and V. Gruhn, Model-Driven Software Development, Springer-Verlag, Berlin, Heidelberg, 2006.

[18] OMG® Unified Modeling Language, OMG, 2017, Available: https://www.omg.org/spec/UML/2.5.1/PDF, Retrieved: 29.08.2020.

[19] T. Ritala and S. Kuikka, UML automation profile: Enhancing the efficiency of software development in the automation industry, IEEE, 2007, Conference proceeding, 2007 5th IEEE International Conference on Industrial Informatics.

[20] G. Kandare, T. Lukman and G. Godena, A New Approach to Control Systems Software Development, Case Studies in Control : Putting Theory to Work, Springer, London, 2013, pp. 363-406.

[21] J. Love, Process Automation Handbook : A Guide to Theory and Practice, Springer, London, 2008.

[22] Zippel Stefan. Process Industry 4.0, 2018, Available: https://www.isa.org/intech/20181202/, Retrieved: 25.4.2020.

[23] C. Tominski, Event-Based Visualization for User-Centered Visual Analysis, Rockstock University, 2006, Thesis.

[24] B. Vogel-Heuser *et al*, Challenges for Software Engineering in Automation, Journal of Software Engineering and Applications, 2014, pp. 440-451.

[25] K. John and M. Tiegelkamp, IEC 61131-3 : Programming Industrial Automation Systems - Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making AIDS, Springer, Berlin, Heidelberg, 2010.

[26] U. Ryssel, H. Dibowski and K. Kabitzsch, Generation of function block based designs using semantic web technologies, IEEE, 2009, Conference proceeding, 2009 IEEE Conference on Emerging Technologies & Factory Automation.

[27] M. Colla, T. Leidi and M. Semo, Design and implementation of industrial automation control systems: A survey, IEEE, 2009, Conference proceeding, 2009 7th IEEE International Conference on Industrial Informatics.

[28] J. Peltola *et al*, Challenges in industrial adoption of model-driven technologies in process control application design, IEEE, 2011, pp. 565-572, Conference proceeding, 2011 9th IEEE International Conference on Industrial Informatics.

[29] B. Vogel-Heuser *et al*, Evolution of software in automated production systems: Challenges and research directions, The Journal of Systems & Software, 2015, pp. 54-84.

[30] Yahya Nazer. ISA 106 - ChemPID Webinar, 2017, Available: https://www.youtube.com/watch?v=gtjIz_K35QM, Retrieved: 25.4.2020.

[31] ISA-106 Procedure Automation for Continuous Process Operations Technical Report 1, Available: https://web-material3.yokogawa.com/ISA_106_TR1_Infographic.us.pdf, Retrieved: 29.08.2020.

[32] Yokogawa. Exapilot Standard Operation Efficiency Improvement Package, Available: https://web-material3.yokogawa.com/GS36J06B10-01E.pdf?_ga=2.58828781.176123816.1593584561-3640551.1592982208, Retrieved: 31.08.2020.

[33] Programmable controllers -- Part 3: Programming languages, IEC 61131-3, 2013.

[34] K. Thramboulidis and G. Frey, Towards a Model-Driven IEC 61131-Based Development Process in Industrial Automation, Journal of Software Engineering and Applications, 2011, pp. 217-226.

[35] Tomáš Bezák, Usage of IEC 61131 and IEC 61499 standards for creating distributed control system Scientific Monographs in Automation and Computer Science, Ilmenau University Library, 2011.

[36] Function blocks - Part 1: Architecture, IEC 61499-1, 2012.

[37] W. W. Dai and V. Vyatkin, A case study on migration from IEC 61131 PLC to IEC 61499 function block control, IEEE, 2009, pp. 79-84, Conference proceeding, 2009 7th IEEE International Conference on Industrial Informatics.

[38] D. O'Sullivan and D. Heffernan, VHDL architecture for IEC 61499 function blocks, IET computers & digital techniques, 2010, pp. 515-524.

[39] A. Zoitl and R. Lewis, Modelling Control Systems using IEC 61499, The Institution of Engineering and Technology, London, 2014.

[40] K. Thramboulidis, IEC 61499 vs. 61131: A Comparison Based on Misperceptions, Journal of software engineering and applications, 2013, pp. 405-415.

[41] J. P. Peltola *et al*, Process control with IEC 61499: Designers' choices at different levels of the application hierarchy, IEEE, 2006, pp. 183-188, Conference proceeding, 2006 4th IEEE International Conference on Industrial Informatics.

[42] Gary L. Pratt. Leverage object-oriented industrial programming , 2019, Available: https://www.controleng.com/articles/leverage-object-oriented-industrial-programming/, Retrieved: 25.4.2020.

[43] SIMATIC PCS 7 Plant Automation Accelerator using a Practical Example, Siemens, 2019.

[44] Control Module (CM) Technology - Efficient Engineering in SIMATIC PCS 7, Siemens, 2019.

[45] SPPA-T3000 System Overview, Siemens, 2008, Available: https://www.siemens.com.tr/i/content/3852_1_T3000-SystemOverview_March2008.pdf, Retrieved: 25.4.2020.

[46] T. Strasser *et al*, A research roadmap for model-driven design of embedded systems for automation components, IEEE, 2009, Conference proceeding, 2009 7th IEEE International Conference on Industrial Informatics.

[47] L. Ferrarini *et al*, Domain specific views in model-driven embedded systems design in industrial automation, IEEE, 2009, pp. 702-707, Conference proceeding, 2009 7th IEEE International Conference on Industrial Informatics.

[48] T. Vepsalainen *et al*, Assessing the industrial applicability and adoption potential of the AUKOTON model driven control application engineering approach, IEEE, 2010, pp. 883-889, Conference proceeding, 2010 8th IEEE International Conference on Industrial Informatics.

[49] D. Hästbacka, T. Vepsäläinen and S. Kuikka, Model-driven development of industrial process control applications, The Journal of Systems & Software, 2011, pp. 1100-1113.

[50] B. Vogel-Heuser and B. Vogel-Heuser, Usability Experiments to Evaluate UML/SysML-Based Model Driven Software Engineering Notations for Logic Control in Manufacturing Automation, Journal of Software Engineering and Applications, 2014, pp. 943.

[51] B. Vogel-Heuser, D. Witsch and U. Katzke, Automatic code generation from a UML model to IEC 61131-3 and system configuration tools, IEEE, 2005, pp. 1034-1039 Vol. 2, Conference proceeding, 2005 International Conference on Control and Automation.

[52] B. Vogel-Heuser *et al*, Usability and benefits of UML for plant automation - some research results, Atp international, 2005.

[53] S. Brinkmann, Qualitative Interviewing, Oxford University Press, Oxford, 2013.

[54] S. Q. Qu and J. Dumay, The qualitative research interview, Qualitative Research in Accounting & Management, 2011, pp. 238-264.